COBB

INSIDE DOS

Tips & techniques for MS-DOS & PC-DOS

April 1993
Vol. 4 No. 4

Capturing the date and time as environment variables

In the past months, we've received several questions about using the date and time in batch files. This month, we'll try to answer some of those questions. We'll begin by showing you how to create TODAY.BAT, a batch file that lets you store in environment variables named DATE and TIME the date and time you run the batch file. In the related article "Labeling Diskettes with the Current Date," on page 4, we'll show you a simple application for TODAY.BAT. Also in this issue, Van Wolverton shows you how to record the time of important events with LOG.BAT.

This article introduces the TODAY.BAT file, which is based on a batch file of the same name presented as a Microsoft technical note. We'll show you how to create TODAY.BAT and explain how it works. We'll also show you the basics of using TODAY.BAT within another batch file.

Creating TODAY.BAT

As with any batch file, you create TODAY.BAT with the DOS Editor or another word processing program that allows you to save files in an ASCII, text-only format. Figure A shows the instructions you need to include in the TODAY.BAT file. Please note that you should place TODAY.BAT in your C:\BATCH directory, which should be on your path. If you decide to put this batch file in another directory, be sure the directory is on your path, and change the path in lines four and five.

Figure A

```
@echo off
rem TODAY.BAT creates DATE and TIME environment variables.
if not "%4"=="" goto :SETDATE
copy /b c:\batch\today.bat+ > nul
dir c:\batch\today.bat | find "TODAY" > tempdate.bat
tempdate
:SETDATE
set date=%3
set time=%4
del tempdate.bat
```

TODAY.BAT creates environment variables that store the current date and time.

Once you've created TODAY.BAT, you'll have a handy batch file utility that you can call from other batch files. Later in this article, we'll show you some techniques for using TODAY.BAT, but first let's review how the batch file works.

How TODAY.BAT works

As with most batch files, TODAY.BAT begins with a statement to turn off ECHO, which prevents DOS from displaying each line of the batch file as it executes the line. The second statement begins with the REM (remark) command and allows you to place a note in the batch file.

The third line in the batch file

```
if not "%4"=="" goto :SETDATE
```

checks to see whether TODAY.BAT has a fourth parameter. As we'll see in a moment, TODAY.BAT will have a fourth parameter only in a special circumstance.

IN THIS ISSUE

Next, TODAY.BAT executes a COPY command:

```
copy /b c:\batch\today.bat+ > nul
```

This command is the "trick" that enables TODAY.BAT to get the current date and time. With this line, TODAY.BAT copies itself—thereby updating the date and time on its directory listing.

In fact, we published a similar technique in the article "A Command for Updating Files' Date and Time Stamps" in the April 1992 issue of *Inside DOS*. As we explained in that article, the /B switch tells the COPY command to treat the file as a binary file rather than a text file. After you type *copy /b*, type the complete path and name for TODAY.BAT. (Remember, if you aren't using the directory C:\BATCH, change the path to the directory you've chosen.) Usually, you'd type a filename after the plus sign, which would tell DOS to combine the files named in the COPY command. However, since you don't want to change TODAY.BAT, you don't specify a file after the plus sign. Normally, this form of the COPY command would accomplish nothing, but the /B switch allows you to update TODAY.BAT without actually adding anything to the batch file. Finally, the direction > *nul* sends the message the COPY command generates to the NUL device, preventing the message from appearing onscreen.

The fifth line uses the DIR and FIND commands to create a temporary batch file:

```
dir c:\batch\today.bat | find "TODAY" > tempdate.bat
```

The DIR command simply generates a directory listing for C:\BATCH\TODAY.BAT. For example, if you ran TODAY.BAT at 10:20 a.m. on April 1, 1993, DOS would generate output similar to this:

```
 Volume in drive C is COBB
 Volume Serial Number is 18EE-83ED
 Directory of C:\BATCH

TODAY    BAT       255      04-01-93    10:20a
         1 file(s)            255 bytes
                          11665498 bytes free
```

Although this directory listing displays the information you need—the date and time the COPY command updated TODAY.BAT—it also displays the volume name and other information you don't need. The FIND command refines the directory listing to the line

```
TODAY    BAT       255      04-01-93 10:20
```

and sends that line to a temporary file named TEMPDATE.BAT.

The next line of the batch file

```
tempdate
```

simply runs the TEMPDATE.BAT file. (Note that we don't use the CALL command to run the temporary batch file and return automatically to TODAY.BAT. In this unusual situation, we want to *quit* the current

INSIDE DOS™

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group.

Prices
Domestic ..... $49/yr. ($6.00 each)
Outside U.S. ..... $69/yr. ($8.50 each)

Phone
Toll free ..... (800) 223-8720
Local ..... (502) 491-1900
Customer Relations Fax ..... (502) 491-8050
Editorial Department Fax ..... (502) 491-4200

Address
You may address tips, special requests, and other correspondence to

The Editor, *Inside DOS*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Postmaster
Second class postage is paid in Louisville, KY. Send address changes to

Inside DOS
P.O. Box 35160
Louisville, KY 40232

Back Issues
To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $6.00 each, $8.50 outside the U.S. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you. Please identify the issue you want by the month and year it was published. Customer Relations can also provide you with an issue-by-issue listing of all the articles that have appeared in *Inside DOS*.

Staff
Editor-in-Chief ..... Suzanne Thornberry
Contributing Editors ..... Van Wolverton
David Reid
Editing ..... Martha A. Clayton
Elizabeth Welch
Production Artist ..... Julie Jefferson
Design ..... Karl Feige
Publications Manager ..... Tara Dickerson
Circulation Manager ..... Brent Shean
Publications Director ..... Linda Baughman
Editorial Director ..... Jeff Yocom
Publisher ..... Douglas Cobb

Advisory Board
Earl Berry Jr.
Tina Covington
Marvin D. Livingood

batch file—TODAY.BAT—before running the second one—TEMPDATE.BAT.)

Since TODAY.BAT executes TEMPDATE.BAT at this point, let's look at how TEMPDATE.BAT works. The file contains a single line:

```
TODAY   BAT    255    04-01-93    10:20a
```

Of course, when DOS runs the TEMPDATE.BAT file, it will expect the first word to be a command. Since the first word of TEMPDATE.BAT is *today*, DOS will run TODAY.BAT—again. But this time, the TODAY.BAT file will include parameters captured from the directory listing. Figure B shows how TEMPDATE.BAT assigns parameters to TODAY.BAT. The symbols in red show the parameters that TEMPDATE.BAT uses when it runs TODAY.BAT.

Figure B

```
        %1     %2      %3          %4

TODAY   BAT    255    04-01-93    10:20a
```

TEMPDATE.BAT runs TODAY.BAT with four parameters. The third and fourth parameters represent the date and time.

Now, let's return to TODAY.BAT to see how running it with these parameters sets the DATE and TIME environment variables. After processing the *@echo off* and *rem* instructions, the TODAY.BAT file tests for the presence of a fourth parameter by using the instruction

```
if not "%4"=="" goto :SETDATE
```

Since TEMPDATE.BAT runs TODAY.BAT with four parameters, this condition is met. That is, "%4" (the fourth parameter) isn't equal to empty quotation marks (""). As a result, TODAY.BAT jumps to the :SETDATE label:

```
:SETDATE
set date=%3
set time=%4
del tempdate.bat
```

As its name implies, the :SETDATE section contains the instructions for setting the environment variable DATE to the third parameter, which is the date shown in the updated directory listing from TODAY.BAT. The next instruction in this section sets the time as the fourth parameter—the time from the directory listing. Finally, the last line of the TODAY.BAT batch file deletes the TEMPDATE.BAT file.

Now, let's look at some basic ways you can use TODAY.BAT.

Using TODAY.BAT

You can run TODAY.BAT from another batch file to store the date and time as environment variables. Just add the line

```
call today
```

to the batch file before it issues a command that needs the DATE and TIME variables. For example, suppose you want your AUTOEXEC.BAT file to display the date and time when you boot up. If you'd like, you can do this by adding the following lines to the end of the AUTOEXEC.BAT file:

```
call today
cls
echo.
echo Today is %date% and the time is %time%.
echo.
```

As you can see, you enclose the DATE and TIME variable names in percent signs when you refer to them in a batch file. (The *echo.* lines simply display blank lines around the message.) When you boot your system at 9:05 a.m. on April 1, 1993, AUTOEXEC.BAT will execute your usual bootup commands, then clear the screen and display the following message:

```
Today is 04-01-93 and the time is 9:05a.
```

Although DOS will let you use the DATE and TIME environment variables within a batch file, you can't use them directly from the DOS prompt. If you type *%date%* or *%time%* at the DOS prompt, DOS will assume you're referring to a file named %DATE% or %TIME%.

Cleaning up the environment

Whenever you run TODAY.BAT, it creates two environment variables. Although the DATE variable is valid all day, the TIME variable is accurate only immediately after you run TODAY.BAT. Because the TIME variable is of limited value, we recommend that you always delete this variable after running TODAY.BAT. You can delete any environment variable by setting it equal to nothing. Returning to our previous example, you can add a line to your AUTOEXEC.BAT file to delete the TIME variable after the ECHO command displays the date and time:

```
call today
cls
echo.
echo Today is %date% and the time is %time%.
echo.
set time=
```

You can also add the line *set date=* to delete the DATE environment variable. However, the DATE environment variable is potentially useful all day.

For example, if you run TODAY.BAT from your AUTOEXEC.BAT file, you can use the DATE environment variable in another batch file without running TODAY.BAT a second time (providing, of course, that you reboot every day).

Note

As with all environment variables, creating the DATE and TIME environment variables uses up some of DOS' environment space. DATE and TIME are relatively small variables, so they probably won't cause you any problem. However, you may receive the following message if DOS runs out of room to store them:

```
Out of environment space
```

If you see this message, edit the CONFIG.SYS file to increase the environment size specified with the /E parameter of the SHELL command. If there's no SHELL command in CONFIG.SYS, add one that looks like this:

```
shell = c:\dos\command.com c:\dos /e:512 /p
```

Once you've added or modified the SHELL command, reboot your PC by pressing [Ctrl][Alt][Delete]. If you still receive the message at 512, increase the environment space from 512 to 640 and try again.

Conclusion

DOS doesn't provide a built-in way of referring to the current date and time. TODAY.BAT gives you a way of storing the date and time as environment variables. The accompanying article, "Labeling Diskettes with the Current Date," shows you a simple application for using TODAY.BAT. On page 6, Van Wolverton shows you a file-based way of working with the date and time in the article "LOG.BAT Helps You Keep Track of Important Events." We hope that you'll be able to adapt these basic techniques for working with the date and time for your own batch files.

BATCH FILE TECHNIQUE

Labeling diskettes with the current date

Most of us don't think much about the LABEL command. You might use it occasionally to put your name or initials on a diskette or perhaps to number a series of diskettes. But there's one use you might have missed: You can use labels for keeping track of the date drafts or file copies originated.

Figure A

```
@echo off
rem DAYLABEL.BAT labels a diskette with today's date.
rem You need TODAY.BAT to use DAYLABEL.BAT.
if "%1"=="" goto :ERROR
call today
label %1: %date%
set time=
set date=
goto :END
:ERROR
echo.
echo Please specify which drive contains the diskette
echo you want to label. For example, to label a diskette
echo in the A: drive, enter:
echo.
echo     daylabel a
echo.
:END
```

DAYLABEL.BAT labels a diskette with the date.

This technique is especially useful if you work in an office where you circulate diskettes with materials for review by several people. By placing the date on the diskette's label, you have a convenient way to keep track of various versions. Dating the diskette is more reliable than dating the files, since others may update the files and make it difficult to figure out the date the copy originated. Better still, the label will appear any time you issue the DIR command.

In this article, we'll show you how to create the DAYLABEL.BAT batch file, shown in Figure A. As you'll see, DAYLABEL.BAT uses TODAY.BAT (see "Capturing the Date and Time as Environment Variables" on page 1) to record the date, then uses the DATE environment variable as input for the LABEL command.

How DAYLABEL.BAT works

DAYLABEL.BAT begins with a statement that turns off ECHO. Then, we've included two optional remark statements: one that reminds you of the batch file's function and another that reminds you to run DAYLABEL.BAT from TODAY.BAT.

The IF statement in line 4

```
if "%1"=="" goto :ERROR
```

checks to see whether you typed a parameter (%1) after *daylabel* when you ran the batch file. If you didn't, the batch file goes to the :ERROR section. This check helps to ensure that you don't unintentionally label your hard disk drive—although you could label that drive by typing *daylabel c*.

The *call today* command on the next line calls the TODAY.BAT file, which sets the DATE and TIME environment variables. (We've assumed that you won't run TODAY.BAT in your AUTOEXEC.BAT file and therefore didn't save the DATE variable.)

The line that follows carries out the LABEL command:

```
label %1: %date%
```

As you can see, this line automatically includes the colon with the drive letter you specify as DAYLABEL.BAT's first parameter. You don't need to type the colon when you run DAYLABEL.BAT. This line also specifies the DATE environment variable as the label name.

The next three lines of the batch file

```
set time=
set date=
goto :END
```

delete the DATE and TIME environment variables and quit the batch file. (If you've decided to keep the DATE variable, you can delete the line *set date=*.)

The section under the :ERROR label will simply present a message if you forget to type a drive letter after *daylabel* when you run the batch file:

```
:ERROR
echo.
echo Please specify which drive contains the diskette
echo you want to label. For example, to label a diskette
echo in the A: drive, enter:
echo.
echo     daylabel a
echo.
:END
```

(You can use tabs or spaces to indent the line *daylabel a*.)

Using DAYLABEL.BAT

Using DAYLABEL.BAT is fairly straightforward: You simply type *daylabel* followed by the letter of the drive you want to label. For example, to label a diskette in the B: drive, enter the command

```
C:\>daylabel b
```

You can check the volume label for the diskette in the B: drive by issuing the DIR command:

```
C:\>dir b:
```

At the top of the directory listing, DOS will display the volume label. For example, if you ran the batch file DAYLABEL.BAT on April 1, 1993, you'd see the following information:

```
Volume in drive B is 04-01-93
```

Below the volume label, DOS displays the volume serial number followed by the directory of files on the B: drive.

DOSKEY TIP

Running a COM, EXE, or BAT file having the same name as a macro

A DOSKEY macro always takes precedence over a command (COM), executable program (EXE), or batch file (BAT) of the same name. For example, the COPY macro that we created in the article "Guarding Against the Hidden DOS Trap," which appeared in the February issue of *Inside DOS*, takes precedence over DOS' own COPY command. When you type *copy* followed by a filename and destination, the COPY macro executes SAFECOPY.BAT instead of the usual DOS command.

But suppose you really want to use the COPY command rather than SAFECOPY.BAT? For example, you might want DOS to replace files of the same name with newer versions. SAFECOPY.BAT won't replace files for you—it just displays an error message.

You can bypass the COPY macro by typing a space before the COPY command. For example, if you want to copy everything in your C:\MEMOS directory to a diskette in the B: drive by using DOS' COPY command rather than your COPY macro, simply type a space before the command:

```
C:\> copy \memos\*.* b:\
```

DOS will copy all the files from the directory to the diskette, even if that means copying over some files that are already on the diskette.

VAN WOLVERTON

LOG.BAT helps you keep track of important events

Do you ever need to keep track of recurring events, such as car trips, telephone calls, or computer use? Maybe it isn't your own curiosity or need to know that you must satisfy; perhaps your employer or the IRS insists on good record keeping. The computer is an excellent tool for such record keeping.

If you want to use a computer for logging events, this article will show how to create a batch file—named LOG.BAT—that automates the process. LOG.BAT lets you maintain a log file of entries, each entry marked with the time and date. For security, LOG.BAT protects the log file against inadvertent change or deletion. And, for flexibility, you can create several log files to keep track of several kinds of events.

In a moment, we'll show you how LOG.BAT works. But first, let's review a few of the principles LOG.BAT is based on.

Redirection is the key

LOG.BAT, which is shown in Figure A, creates and updates the log file by using two redirection symbols and the COPY command:

- The > symbol redirects the output of a command by creating a new file to receive the output. If a file exists with the specified name, the > symbol tells DOS to overwrite the original file.
- The >> symbol redirects the output of a command by adding the output to the file if it exists or by creating the file if it doesn't exist.
- The COPY command combines files if you separate their filenames with a plus sign. LOG.BAT uses the COPY command to copy the console (CON) to the log file. CON is the device name that DOS uses to refer to the keyboard (input) and display (output).

Redirecting command input

To put the date and time into the log file, LOG.BAT uses the DATE and TIME commands. Normally, the DATE and TIME commands halt your system until you press the [Enter] key. But LOG.BAT redirects the input of the DATE and TIME commands from a file named ENTER, which contains nothing but a carriage return. In effect, the ENTER file allows LOG.BAT to respond to the prompt that the DATE and TIME commands issue.

Figure A

```
@echo off
if not "%1"=="" goto :OK
echo.
echo You must enter the name of a file as a
echo parameter (for example, LOG FONE.TXT).
echo If the file doesn't exist, it is created.
echo.
goto :END
:OK
cls
if exist %1 goto :EXISTS
echo.
echo Type file description, then
echo press [F6] and [ENTER].
copy con %1 > nul
echo.
:EXISTS
attrib -r %1
echo.
echo Type the entries to be added to %1.
echo After the last line, press [F6] and [ENTER].
echo.
echo. >> %1
echo ------------------------------ >> %1
date < c:\batch\enter ¦ find "C" >> %1
time < c:\batch\enter ¦ find "C" >> %1
echo ------------------------------ >> %1
copy %1+con %1 > nul
echo. >> %1
attrib +r %1
:END
```

LOG.BAT creates and maintains a file of dated entries.

Then, LOG.BAT redirects the output of the DATE and TIME commands to the FIND filter with the parameter *"C"*, yielding just the second lines of the commands' output (*Current date is...* and *Current time is...*). Before you can use LOG.BAT, you'll need to create the ENTER file. We suggest that you create this file in your C:\BATCH directory. To do so, you first enter the following COPY command:

```
C:\>copy con c:\batch\enter
```

When you press [Enter], the COPY command will move the cursor to the line below the command and wait for you to type the contents of the file. Since the ENTER file contains only a carriage return, you press the [Enter] key a second time. The COPY command will scroll down one line. Next, you press the [F6] key

followed by [Enter] to quit the COPY command. When you've finished, you'll see the following exchange with DOS:

```
C:\>copy con c:\batch\enter   (You press [ENTER])
                              (You press [ENTER])
^Z                            (You press [F6], then [ENTER])

    1 files(s) copied
```

How LOG.BAT works

The first line of LOG.BAT simply prevents DOS from echoing the lines of the batch file to the screen. The second line

```
if not "%1"=="" goto :OK
```

tests to see whether you entered a parameter after *log* when you ran the batch file. This first parameter should be the name of the log file. If you didn't include a parameter, the ECHO statements that follow display a message reminding you to enter a filename. Then, the batch file quits by jumping to the :END label. (The *echo.* lines simply display blank lines around the message.)

On the other hand, if you *did* enter a parameter, LOG.BAT jumps to the :OK section:

```
:OK
cls
if exist %1 goto :EXISTS
echo.
echo Type file description, then
echo press [F6] and [ENTER].
copy con %1 > nul
echo.
```

This section checks to see whether the name of the file you entered as the first parameter (%1) exists. If the file exists, LOG.BAT jumps to the :EXISTS label. If not, the batch file displays the message *Type file description, then press [F6] and [ENTER].* As you may recall from creating the ENTER file, [F6] and [Enter] are the keystrokes that you use when copying text from the CON (console) device. So, after you press [F6] and [Enter], LOG.BAT will execute the statement *copy con %1 > nul*. This statement tells DOS to copy the file description that you typed from the console to the file you entered as the first parameter (%1). Redirecting the output of this COPY command to the NUL device just prevents DOS from displaying the message *1 file(s) copied*.

Whether you just created the log file or you specified a filename that already exists, LOG.BAT will execute the :EXISTS section when you create an entry for your log file. The first command in that section

```
attrib -r %1
```

turns off the read-only attribute of the log file. (The new file won't have the attribute set the first time you create an entry, but you'll need this ATTRIB command for subsequent entries.) Then, the ECHO commands that follow display a blank line, instructions to type the entries to be added to the log file, and another blank line:

```
echo.
echo Type the entries to be added to %1.
echo After the last line, press [F6] and [ENTER].
echo.
```

After you've typed the entry and pressed [F6] and [Enter], the next commands send information to the log file you entered as the first parameter:

```
echo. >> %1
echo ---------------------------- >> %1
date < enter ¦ find "C" >> %1
time < enter ¦ find "C" >> %1
echo ---------------------------- >> %1
```

The first of these ECHO statements simply sends a blank line to the log file; the next one sends a dashed line, created by pressing the hyphen repeatedly. The next two lines apply the redirection principles we described earlier. First, the DATE command gets input—a carriage return—from the ENTER file, allowing DOS to execute the DATE command with no input from the keyboard. The DATE command would produce output like this:

```
Current date is Fri 04-02-1993
ENTER new date (mm-dd-yy):
```

Obviously, you don't need the lines *ENTER new date (mm-dd-yy):* cluttering up your log file. So the FIND command retrieves only the line containing a "C"—the line *Current date is Fri 04-02-1993*. Finally, the >> redirection symbol tells DOS to append the found line to the log file. The TIME command works in the same way, sending the line *Current time is...* to the log file. After sending the date and time to the log file, an ECHO statement again sends a dashed line to the file.

Now that LOG.BAT has sent the date and time to the log file, it must send the entry you typed to the file, too. It does so with a COPY command:

```
copy %1+con %1 > nul
```

This COPY command adds the lines you type by using the plus sign to combine the log file (represented by %1) with the entry you typed at the console (CON) into the log file (%1, again). Here again, LOG.BAT redirects the output of the COPY command to NUL to suppress the

1 file(s) copied message. Next, the *echo.* command sends another blank line to the log file. Finally, the command

```
attrib +r %1
```

turns on the read-only attribute so that you don't accidentally change or delete the log file. The :END label that follows simply marks the end of the batch file.

Logging events with LOG.BAT

To start a log file, you just pick a filename and type it as the parameter of the command. For example, suppose you wanted to keep track of telephone calls in a file named C:\DOCS\PHONE.LOG and you've already changed to the C:\DOCS directory. When you type the first entry, the batch file prompts you to enter a file description:

```
C:\DOCS>log phone.log
Type file description, then
press [F6] and [ENTER].
```

Type *April Phone Log*, then press [F6] and [Enter]. Now LOG.BAT prompts you to type the log entry:

```
Type the entries to be added to phone.log.
After the last line, press [F6] and [ENTER].
```

Type *Called realtor about Oak Street building* and then press [F6] and [Enter].

The next time you use the LOG command, the batch file prompts you to type just the next entry because the file already exists:

```
C:\>log phone.log

Type the entries to be added to phone.log.
After the last line, press [F6] and [ENTER].
```

What if you make two calls in a row? Type *Called Robb, Cheatham, and Steele about depositions*, press [Enter], type *Called home*, then press [F6] and [Enter]. At this point, here's what your PHONE.LOG file would contain:

```
April Phone Log
------------------------------
Current date is Tue 04-27-1993
Current time is  9:16:44.47a
------------------------------
Called realtor about Oak Street building
------------------------------
Current date is Tue 04-27-1993
Current time is  9:27:02.76a
------------------------------
Called Robb, Cheatham, and Steele about depositions
Called home
```

Working with the log file

The log file is an ordinary text file that you can edit or print just as you would any other text file. If you plan to change or delete the log file, you must first turn off the read-only attribute by using the ATTRIB command. If you named the log file PHONE.LOG, for example, you'd type

```
C:\>attrib -r phone.log
```

Once you've issued this command, you'll be able to open the PHONE.LOG file with a text editor (such as DOS EDIT) or a word processor (such as Microsoft Word).

To simplify things

You could make LOG.BAT easier to use by specifying the name of the log file in the batch file, eliminating the need to type the name of the log file each time you use the command. However, doing so makes LOG.BAT less flexible because you can log only one type of event. If you intended to log only telephone calls, for example, you might name the log file C:\DOCS\PHONE.LOG. To make this change, you delete lines from the second line through the :OK label (since you won't need to enter a parameter when you run this version of LOG.BAT) and replace all occurrences of the *%1* parameter with *C:\DOCS\PHONE.LOG*. Now, whenever you want to run the batch file, you simply enter

```
C:\>log
```

Even if you keep several log files, you can simplify things somewhat by keeping them in the same directory and naming all with the same extension. For example, you might keep all your log files in the C:\DOCS directory and use the LOG extension. You could add those constants to the *%1* parameter in the batch file. Just change each occurrence of *%1* to *C:\DOCS\%1.LOG*. With this change, you can run LOG.BAT from any directory simply by typing *log* followed by the root name of the log file. Returning to our example, you could update your phone log by entering

```
C:\>log phone
```

The efficiency of batch files like LOG.BAT is one reason many of us continue to use DOS, even in the face of pretty advances like Windows and OS/2. These little batch file tools aren't especially elegant, but they're right in the spirit of *personal* computing: They let us tailor the computer to our needs with only a minimum of effort and expense.

Contributing Editor Van Wolverton is the author of the best-selling books Running MS-DOS *and* Supercharging MS-DOS. *Van, who has worked for IBM and Intel, currently lives in Alberton, Montana.*

Adding a few lines to your batch files gives them built-in troubleshooting

Chris Wolcott of Pensacola, Florida, sent us this technique for debugging batch files.

Few of us are immediately satisfied with our first draft of a new batch file. Often, we have to track down a minor typographical error or rearrange the lines to get the batch file to work as we intend. In the article "Removing *@echo off* Helps You Debug Batch Files" in the July 1992 issue of *Inside DOS*, we showed you how removing the *@echo off* statement allows you to see each command as the batch file issues it—a valuable technique when you're troubleshooting a batch file.

Chris Wolcott sent us a series of batch file statements that takes the debugging technique a step further. He places the code shown in Figure A at the beginning of every batch file he writes. Then, if he needs to debug the batch file, he simply types the batch file's name followed by *dbug*. (On our computers running on a 386 or 486 chip, adding the code didn't seem to slow down our batch files. However, you might notice the difference on a 286-, 8088-, or 8086-based system.)

Figure A

```
@echo off
for %%a in (dbug DBUG) do if "%1"=="%%a" goto :DBUG
goto :MAIN
:DBUG
shift
echo on
:MAIN
```

Placing these seven lines at the top of your batch file will let you troubleshoot it if anything goes wrong.

Let's look briefly at how these instructions work. The first line, *@echo off*, simply tells DOS not to display the batch file commands as it issues them. (This should be the *only* @echo off statement in your batch file.) The next line is a FOR statement:

```
for %%a in (dbug DBUG)  do if "%1"="%%a" goto :DBUG
```

This FOR statement checks to see whether you typed *dbug* or *DBUG* after the batch file's name when you ran the batch file. To do so, the FOR command uses the batch file variable *%%a*. When you run a batch file containing this code segment, it will replace *%%a* with each value in the set *(dbug DBUG)*. By placing both *dbug* and *DBUG* in the set, you can enter either the lower- or uppercase form when you want to troubleshoot the batch file.

The second part of the FOR statement tells DOS what to do with each value in the set. Immediately after the DO command, the statement

```
if "%1"="%%a"
```

checks the value of %1, which represents the first parameter you typed after the filename when you ran the batch file. The FOR statement first replaces *%%a* with *dbug*, then with *DBUG*.

To get an idea of how this process works, let's suppose you place the lines shown in Figure A at the beginning of the batch file DAYLABEL.BAT, which appears on page 4. If you ran DAYLABEL.BAT with the command

```
C:\BATCH\>daylabel dbug
```

the FOR command would find that *%1* is equal to *%%a* when it compares the values. Since

```
"dbug"="dbug"
```

the batch file will jump to the :DBUG label.

Similarly, if you run the batch file by entering

```
C:\>DAYLABEL DBUG
```

the batch file will find, when it processes the second value in the set *(dbug DBUG)*, that

```
"DBUG"="DBUG"
```

and go to the :DBUG label.

On the other hand, if you don't enter either *DBUG* or *dbug* after the batch file's name, the batch file will execute the next statement:

```
goto :MAIN
```

When you add Mr. Wolcott's code to a batch file, you place the :MAIN label right before the statements that make up the body of your batch file. For example, you could place the instructions for DAYLABEL.BAT under the :MAIN label. (*Don't* include a second *@echo off* statement in the MAIN section since the first line of the troubleshooting instructions shown in Figure A turns off ECHO anyway. If you do, the troubleshooting code won't work when you try to use it.)

The SHIFT command in the :DBUG section tells DOS to shift the parameters you typed after the batch file

name. This SHIFT command must precede the *@echo off* line so that your batch file can accept any parameters you want it to use, even if you enter *DBUG* as the first parameter when you run the batch file. For example, with DAYLABEL.BAT, you could troubleshoot the labeling process by using a diskette in the A: drive:

```
C:\BATCH>daylabel dbug a
```

After processing the :DBUG section of code, the batch file would shift parameters so that *a* became the first parameter for the :MAIN section of DAYLABEL.BAT.

An easy way of including the instructions

The troubleshooting instructions are longer than many batch files and might seem inconvenient to type every time you write a batch file. However, you really need to type these instructions only once. As always, when you type them, use Edlin or an ASCII-compatible word processor. When you've finished typing the instructions, take one last look to make sure you haven't made a typo, then save the file as DBUG.TXT. Now, whenever you want to create a new batch file, you can include the troubleshooting instructions by issuing the command

```
C:\BATCH>type dbug.txt > newfile.bat
```

Of course, you'd substitute the name of the batch file you're creating for *newfile.bat*. When you open your new batch file to edit it, you'll find that it begins with the troubleshooting instructions contained in DBUG.TXT. Then, you can begin typing the instructions for your new batch file after the :MAIN label.

To copy the troubleshooting instructions to an existing batch file, you can select lines 2 through 6 in the DOS Editor by pressing [Shift]↓ or by clicking and dragging the mouse over the text. Once you've selected the text, you can press [Ctrl][Insert] to copy the instructions. Then, open the existing batch file by issuing the Open... command from the File menu ([Alt]F,O). Move the cursor to the second line of your batch file—the one after *@echo off*—then press [Shift][Insert] to paste the instructions into the batch file.

WINDOWS TIP

Setting a special prompt for DOS sessions run from Windows

Thanks to Mike O'Mara, editor-in-chief of Windows Report, *a biweekly newsletter from The Cobb Group, for submitting this tip.*

If you sometimes work with Microsoft Windows, you've probably discovered you can run a DOS session by double-clicking on the MS-DOS Prompt icon. (Windows installs this MS-DOS Prompt icon in the Main program group, but you may have moved it from its default location.) If you're accustomed to working with DOS, it's easy to forget that you're really working under a Windows session—especially if Windows' reminder message scrolls off the DOS screen.

When you use the MS-DOS prompt option, Windows will remain resident in your system's memory. You can enter commands as you usually do from the DOS prompt. But, if you try to run a second copy of Windows by entering *win* at the prompt, Windows will present a warning message. Unfortunately, Windows won't warn you when you try to run a non-Windows application. If you accidentally run a large DOS-based application while Windows is still resident, you may encounter some unusual memory-related problems.

Fortunately, Windows provides an easy way of reminding yourself that you've only temporarily exited from Windows: You simply set the WINPMT environment variable to include a message about returning to Windows. The WINPMT message will replace your usual DOS prompt when you run DOS under Windows. You can define this environment variable through your AUTOEXEC.BAT file.

For example, suppose you want your prompt to say (*Type "EXIT" to return to Windows*) and then display the standard path ($p) and greater than ($g) signs. You can set up this prompt by adding the following line to your AUTOEXEC.BAT file:

```
set winpmt=(Type "EXIT" to return to Windows) $p$g
```

When you reboot or run the AUTOEXEC.BAT file, this SET command will define the WINPMT environment variable. When you select the MS-DOS Prompt icon from Windows, you'll see this more informative prompt:

```
(Type "EXIT" to return to Windows) C:\WINDOWS>_
```

With this prompt, you aren't likely to forget you're running under a Windows session.

Defragmenting files with the CHKDSK command

In the article "Treat Your Hard Disk with Respect—and Care" in the February 1992 issue of *Inside DOS*, you described two options for defragmenting files. The first is backing up the entire hard disk, reformatting it, and restoring all the files. The other option is using a disk optimizer program to defragment and optimize the programs on the hard disk.

I use another method, which is based on DOS' CHKDSK command. Although this CHKDSK method is somewhat time-consuming, it does defragment files, and it doesn't require purchasing additional software.

I work with one directory at a time, issuing the CHKDSK command followed by the path of the directory, a backslash, and the *.* wildcard file specification. For example, to check the C:\DOCS directory, I'd enter the command

```
C:\>chkdsk \docs\*.*
```

CHKDSK will report the usual information about the hard disk (space available and allocation units) and summarize available memory. But the *.* specification causes CHKDSK also to report any fragmented files in the directory. For example, CHKDSK might report that the following document files are fragmented:

```
C:\DOS\HILLTOP.DOC Contains 4 non-contiguous blocks
C:\DOS\SYCAMORE.DOC Contains 7 non-contiguous blocks
```

Now that I know which files are fragmented, I can copy them to a diskette. Pressing the [F3] key will display the previous command. To copy another file from the same directory, I just press [F3] and then the left arrow to delete the name of the previous file. Then I type the name of the next file I want to copy. I'd issue the following commands to copy the fragmented files from the C:\DOCS directory to a diskette in the A: drive:

```
C:\>copy \docs\hilltop.doc a:
C:\>copy \docs\sycamore.doc a:
```

After I've copied all the fragmented files from the directory, I change to the diskette drive and copy the files back to the directory on my hard disk. In effect, this step replaces the old version with an identical but contiguous copy. [*Editor's note*: If you've created a COPY macro that replaces the COPY command with SAFECOPY.BAT, type a space before the COPY command. For more information on overriding a DOSKEY macro, see the article "Running a COM, EXE, or BAT File Having the Same Name as a Macro" on page 5.] Again, I'd enter the following commands to copy the formerly fragmented files back to the C:\DOCS directory:

```
A:\>copy hilltop.doc c:\docs
A:\>copy sycamore.doc c:\docs
```

Because I'm a skeptic, I enter *chkdsk \pathname**.* again to ensure that the files are contiguous.

Throughout this process, I can use a separate diskette for each directory I defragment. Or, I can simply format the diskette when I've finished copying files back to a directory. For example, to quickly format a diskette in the A: drive, I'd enter

```
C:\>format a: /q
```

Then, I can move to the next directory and copy its files to this freshly formatted diskette.

It's also helpful to press [Shift][PrintScreen] to capture a long list of fragmented files that I need to copy.

Marilyn LeRoux
Plattsburgh, New York

Ms. LeRoux's method of defragmenting files is economical and effective. We'd like to add just a few notes to her thorough explanation of the technique.

First, as a precaution, we suggest you run CHKDSK without a file specification to get an idea of the general condition of your hard disk. If you see a lot of bad clusters reported, you may have more serious problems than file fragmentation. You might want to check out your disk with a third-party utilities package, such as The Norton Utilities or PC Tools.

Once you're satisfied that your hard disk is in good condition, you can begin using Ms. LeRoux's technique of identifying and recopying fragmented files. As you go through this process, you may find that some fragmented files generate *Access denied* messages when you try to copy them back to your hard disk. For example, MIRROR.FIL is a very important file that DOS uses to save information for undeleting files. DOS protects this file with the read-only (R) attribute. Although you could use the ATTRIB command to turn off the attribute, copy the file to a diskette, copy it back to your hard disk, and reset the attribute, we recommend you simply leave this file—and other program or system files—alone.

Since most fragmentation occurs on files that are saved and re-edited, you might want to target just directories containing data files. For example, you could target the directories containing your documents, spreadsheets, or database files.

We should also note that some disk optimization programs provide additional services. For example, some of these programs truly "optimize" your hard disk by matching file size to the gaps that occur on your hard disk when you delete a file. Disk optimization programs can do so because they analyze your hard disk *before* moving files around.

Deleting a group of DOS Shell's batch files

The DOS Shell seems to have created hundreds of batch files on my hard disk. Looking at a printout of my C:\DOS directory, I noticed that most of the batch files' names end with DOSCM.BAT or DOSC.BAT. Here's a partial listing of the files:

```
9A2DOSCM   BAT        10    11-15-91     7:22p
49DDDOSC   BAT        16    06-08-92     1:17p
509ADOSC   BAT        16    06-08-92    12:18p
5560DOSC   BAT        24    06-08-92    12:19p
29EDDOSC   BAT        16    07-09-92    10:09a
```

Shouldn't there be something in DOS 5 that kills these batch files once I exit from the DOS Shell? Any ideas before I have a hard disk drive full of one-time batch files?

Charles Schermerhorn
Lompoc, California

We continue to receive a stream of letters from readers who have problems similar to Mr. Schermerhorn's. Usually, the DOS Shell will delete these temporary batch files when you exit the DOS Shell by choosing the Exit... command from the File menu ([Alt]F,X). (If you're using Task Swapper, you'll first have to quit the programs that you've enabled with Task Swapper.)

You can delete the batch files by changing to the directory that contains the files and entering two DELETE (DEL) commands at the DOS prompt. Do *not* try deleting the files when the DOS Shell is running; you should either exit the DOS Shell or reboot your system without running the Shell.

The DELETE commands use the single-character ? wildcard to represent the miscellaneous characters at the beginning of the filenames. So, in order to delete the files from your C:\DOS directory, you simply issue the commands

```
C:\DOS>del ???doscm.bat
C:\DOS>del ????dosc.bat
```

A long-term solution to the problem is to specify a valid temporary directory in your AUTOEXEC.BAT file. If you don't already have one, create a directory named C:\TEMP. Then, you add the following line to your AUTOEXEC.BAT file:

```
set temp=c:\temp
```

Be careful to type the exact name of the directory. Don't leave any extra spaces after the directory name. After you've rebooted your system, the DOS Shell (and many other applications) will place temporary files in the C:\TEMP directory.

Once you've created the temporary directory, it will be easier to monitor and delete any unneeded batch files. But, as we said, you can cut down on the number of these batch files by exiting the DOS Shell before you reboot or turn off your computer.

Microsoft to release MS-DOS 6 this Spring

Beginning with the May 1993 issue, *Inside DOS* will cover DOS 5 and DOS 6. Most of the techniques we'll feature will work under both versions of the operating system. If we describe a technique that takes advantage of DOS 6's new features, we'll try to show you a way to accomplish the same thing with DOS 5.

Next month, we'll show you what's new in DOS 6, Microsoft's latest version of the operating system. We'll give you an overview of the new utilities and memory-management capabilities, which should help you decide whether you want to upgrade.

As always, you'll help us determine which topics we should cover. Whether you're extending DOS 5 or discovering new tricks with DOS 6, we encourage you to send us your suggestions and tips.